Exercice 1 : Arbre parfait

Dans cet exercice, on considère les arbres binaires, définis en OCaml au moyen du type ci-dessous.

```
type 'a btree =
  | V
  | N of 'a * 'a btree * 'a btree
```

On dit d'un arbre binaire qu'il est parfait si toutes les "rangées" de l'arbre sont pleines.

Q. 1 Définir une fonction `est_parfait : 'a btree -> bool` permettant de tester si un arbre binaire est parfait. On pourra proposer un algorithme de complexité $\mathcal{O}(n^2)$ dans un premier temps, avant d'en proposer un de complexité $\mathcal{O}(n)$.

On dit d'un arbre binaire qu'il est complet si toutes les "rangées" de l'arbre sont pleines, sauf éventuellement la dernière, qui doit être complétées de gauche à droite.

Q. 2 Définir une fonction `est_complet : 'a btree -> bool` permettant de tester si un arbre binaire est complet. *Indication :* Ne pas hésiter à demander une indication.

Exercice 2 : Arbre binaire de recherche et préfixe

Dans cet exercice, on considère les arbres binaires, définis en OCaml au moyen du type ci-dessous.

```
type 'a btree =
  | V
  | N of 'a * 'a btree * 'a btree
```

On considère un arbre binaire de recherche dont les étiquettes sont deux à deux disjointes. On appelle chemin d'une étiquette e l'unique chemin menant de la racine au nœud d'étiquette e. Un chemin dans un arbre est représenté au moyen d'une liste de booléens (`false` représente le fait d'aller à gauche, `true` représente le fait d'aller à droite)

Q. 1 Définir une fonction `prefixe : 'a btree -> 'a -> 'a -> bool list` prenant en paramètres un arbre binaire de recherche, deux étiquettes x et y distinctes de cet arbre et retournant le plus long préfixe commun aux chemins de x et de y.

Exercice 3 : Enracinement

Dans cet exercice, on considère des arbres binaires de recherche, définis en OCaml au moyen du type ci-dessous.

```
type 'a btree =
  | V
  | N of 'a * 'a btree * 'a btree
```

Q. 1 Définir une fonction `enracine : 'a btree -> 'a -> 'a btree` prenant en paramètres un arbre binaire de recherche b, une étiquette x de cet arbre et retournant un arbre binaire de recherche contenant exactement les étiquettes de b et ayant x comme racine.

Exercice 4 : Encadrement d'arbres

Dans cet exercice, on considère les arbres binaires, définis en OCaml au moyen du type ci-dessous.

```
type 'a btree =
  | V
  | N of 'a * 'a btree * 'a btree
```

Q. 1 Définir une fonction `encadre : int btree -> int -> int * int` prenant en paramètres un arbre binaire de recherche b, une valeur x et retournant le plus petit encadrement de x (au sens large) par deux entiers se trouvant dans l'ensemble représenté par b. S'il n'est pas possible de majorer (resp. minorer) x on utilisera les valeurs spéciales `min_int` et `max_int`.

Exercice 5 : Liste de tableaux

On considère une représentation des ensembles finis d'entiers au moyen d'une liste de tableaux de booléens. Une telle liste sera de la forme : `[`$\mathtt{t}_{p-1}$`; `$\mathtt{t}_{p-2}$`; ...; `$\mathtt{t}_1$`; `$\mathtt{t}_0$`]` où le tableau $\mathtt{t}_i$ est un tableau de 2^i booléens indiquant dans sa case $j \in [\![0, 2^{i-1}]\!]$ si oui ou non l'élément $2^i - 1 + j$ est, ou non dans l'ensemble représenté. Ainsi l'exemple `ex` ci-dessous représente l'ensemble d'entiers : $\{0, 2, 4, 5, 6, 8, 9, 14\}$.

```
let ex =
  (*  7      8      9      10     11     12     13     14 *)
  [[|false; true ; true ; false; false; false; false; true |];
   (* 3      4      5      6  *)
   [|false; true ; true ; true |];
   (* 1      2    *)
   [|false; true  |];
   (* 0   *)
   [|true |];
   ]
```

Q. 1 Définir une fonction `mem (i: int) (s: bool array list): bool` permettant de tester si un élément `i` est dans l'ensemble `s` représenté.

Q. 2 Définir une fonction `ajout (i: int) (s: bool array list): bool array list` permettant l'ajout d'un élément `i` à l'ensemble `s`. Par exemple l'appel `ajout 5 [[|true|]]` devra produire `[[|false; false; true; false|]; [|false; false|]; [|true|]]`.